Machine Learning, 57, 205–232, 2004

Kernels and Distances for Structured Data

THOMAS GÄRTNER thomas.gaertner@ais.fraunhofer.de
Fraunhofer Institut Autonome Intelligente Systeme, Germany; Department of Computer Science, University of Bristol, United Kingdom; Department of Computer Science III, University of Bonn, Germany

JOHN W. LLOYD jwl@csl.anu.edu.au
Research School of Information Sciences and Engineering, The Australian National University

PETER A. FLACH peter.flach@bristol.ac.uk
Machine Learning, Department of Computer Science, University of Bristol, United Kingdom

Editor: Stan Matwin

Abstract. This paper brings together two strands of machine learning of increasing importance: kernel methods and highly structured data. We propose a general method for constructing a kernel following the syntactic structure of the data, as defined by its type signature in a higher-order logic. Our main theoretical result is the positive definiteness of any kernel thus defined. We report encouraging experimental results on a range of real-world data sets. By converting our kernel to a distance pseudo-metric for 1-nearest neighbour, we were able to improve the best accuracy from the literature on the Diterpene data set by more than 10%.

Keywords: kernel methods, structured data, inductive logic programming, higher-order logic, instance-based learning

1. Introduction

This paper brings together two strands of machine learning of increasing importance: kernel methods and highly structured data. To start with the latter, most real-world data has no natural representation as a single table. In order to apply traditional data mining methods to structured data, extensive pre-processing has to be performed. Research in inductive logic programming (ILP) and multi-relational data mining (Džeroski and Lavrač, 2001) aims to reduce these pre-processing efforts by considering learning from multi-relational data representations directly.

Support vector machines (Boser, Guyon, and Vapnik, 1992; Vapnik, 1995) are a popular recent development within the machine learning and data mining communities. Along with some other learning algorithms like Gaussian processes and kernel principal component analysis, they form the class of kernel methods (Müller et al., 2001; Schölkopf and Smola, 2002). The computational attractiveness of kernel methods comes from the fact that they can be applied in high-dimensional feature spaces without suffering the high cost of explicitly

computing the mapped data. The *kernel trick* is to define a positive definite kernel on the instance space. For such functions it is known that there exists an embedding of the instance space in a linear space such that the kernel on pairs of instances corresponds to the inner product in this space.

Representation is a key issue in bringing together kernel methods and learning from structured data. While the inductive logic programming community has traditionally used logic programs to represent structured data, the scope has gradually been extended and now includes other knowledge representation languages. If we want to devise a kernel function for structured data that generalises those kernel functions that have successfully been applied to attribute-value problems, it is useful to have a representation for structured data that is close to the attribute-value representation. We thus represent individuals as (closed) terms in a typed higher-order logic (Lloyd, 2003). The typed syntax is important for pruning search spaces and for modelling as closely as possible the semantics of the data in a human- and machine-readable form. The individuals-as-terms representation is a natural generalisation of the attribute-value representation and collects all information about an individual in a single term.

The outline of the paper is as follows. Section 2 introduces kernel methods and defines what is meant by 'valid' kernels. Section 3 gives an account of our knowledge representation formalism, which is a typed higher-order logic. Section 4 defines a kernel on the terms of this logic, investigates some theoretical properties, and describes how these kernels can be adapted to particular domains. Section 5 reviews how this kernel can be used to define a distance function on basic terms that satisfies the metric conditions. Section 6 provides a variety of experimental evaluations of this kernel and distance. Section 7 concludes the paper.

2. Kernel methods

In this section we give a brief overview of kernel methods, and review recent work on kernels for data structures such as strings and trees.

Two components of kernel methods have to be distinguished: the kernel machine and the kernel function. Different kernel machines tackle different learning tasks, e.g., support vector machines for supervised learning, support vector clustering (Ben-Hur et al., 2001) for unsupervised learning, and kernel principal component analysis (Schölkopf, Smola, and Müller, 1999; Schölkopf and Smola, 2002) for feature extraction. While the kernel machine encapsulates the learning task and the way in which a solution is sought, the kernel function encapsulates the hypothesis language, i.e., how the set of possible solutions is made up. Different kernel functions implement different hypothesis spaces or even different knowledge representations.

In this section we use the notation $f(\cdot)$ to refer to a function and the notation $f(x)$ to refer to the value of the function $f(\cdot)$ at x. Similarly, $k(\cdot, \cdot)$ is a function with two arguments, while $k(x, \cdot)$ is a function with one (free) argument, and $k(x, x')$ is the value of the function $k(\cdot, \cdot)$ at (x, x').

2.1. Valid kernels

Usually, learning algorithms can be applied in a high-dimensional feature space by transforming the data with a map ϕ. The computational attractiveness of kernel methods comes from the fact that they do not rely on the explicit construction of the images of instances under the feature map but only on the computation of inner products of pairs of instances in the feature space. Often a closed form of the inner product of the mapped data exists, and instead of performing the expensive transformation step ϕ explicitly, a kernel $k(x, x') = \langle \phi(x), \phi(x') \rangle$ calculating the inner product directly can be used.

Whether, for a given function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, a feature transformation $\phi : \mathcal{X} \to \mathcal{H}$ into the Hilbert space $\mathcal{H}$ exists, such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$ for all $x, x' \in \mathcal{X}$, can be checked by verifying that k is positive definite (Aronszajn, 1950). This means that any set, whether a linear space or not, that admits a positive definite kernel can be embedded into a linear space. Thus, throughout the paper, we take 'valid' to mean 'positive definite'. Here then is the definition of a positive definite kernel. ($\mathbb{Z}^+$ is the set of positive integers.)

Definition 2.1 (Positive definite kernel functions). Let $\mathcal{X}$ be a set. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *positive definite kernel* on $\mathcal{X}$ if, for all $n \in \mathbb{Z}^+$, $x_1, \ldots, x_n \in \mathcal{X}$, and $c_1, \ldots, c_n \in \mathbb{R}$, it follows that

$$\sum_{i,j \in \{1,\ldots,n\}} c_i c_j k(x_i, x_j) \geq 0$$

Note that, for a positive definite kernel, the left-hand side of this inequality corresponds to the squared length of a linear combination of vectors in feature space, i.e., $\sum_{i,j} c_i c_j k(x_i, x_j) = \| \sum_i c_i \phi(x_i) \|^2$.

While it is not always easy to prove positive definiteness for a given kernel, positive definite kernels do have some nice closure properties. In particular, they are closed under sum, direct sum, multiplication by a scalar, product, tensor product, zero extension, pointwise limits, and exponentiation (Cristianini and Shawe-Taylor, 2000; Haussler, 1999).

It should be noted that, for a kernel method to perform well on a domain, positive definiteness of the kernel is not the only issue. While there is always a valid kernel that performs poorly (e.g., $k_0(x, x') = 0$), there is also always a valid kernel that performs ideally ($k_c(x, x') = c(x)c(x')$, where $c(z)$ is $+1$ if z is is a member of the concept and -1 otherwise). The suitability of a kernel must ultimately be verified by experiments.

2.2. Kernel machines

The usual supervised learning model (Vapnik, 1995) considers a set $\mathcal{X}$ of individuals and a set $\mathcal{Y}$ of labels, such that the relation between individuals and labels is a probability measure on the set $\mathcal{X} \times \mathcal{Y}$. A basic and well-known approach to supervised learning is to use for

prediction the hypothesis function that minimises the empirical risk (training error)

$$\min_{f(\cdot)\in\mathcal{F}} \frac{1}{n}\sum_{i=1}^{n} V(y_i, f(x_i))$$

where $\mathcal{F}$ is a set of functions, the hypothesis space, and $V : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a function that takes on small values whenever $f(x)$ is a good guess for y and large values whenever it is a bad guess, the loss function.

In practice, it may not be sufficient to minimise the empirical error. It is often necessary to make sure that the functions considered as solutions are stable with respect to small perturbations of the training data. One approach to this problem is known as Tikhonov regularisation (Tikhonov and Arsenin, 1977), which means that we add a regularisation term to the empirical risk. In kernel methods it is common to assume that the hypothesis space forms a Hilbert space and to use the norm of $f(\cdot)$ in the corresponding Hilbert space $\|f(\cdot)\|_{\mathcal{H}}$ as the regularisation term. This gives rise to the corresponding optimisation problem

$$\min_{f(\cdot)\in\mathcal{H}} \frac{C}{n}\sum_{i=1}^{n} V(y_i, f(x_i)) + \|f(\cdot)\|_{\mathcal{H}}^2$$

Different kernel methods arise from using different loss functions, see also Evgeniou, Pontil, and Poggio (2000). Support vector machines, for example, arise from using the so-called hinge loss $V(y, f(x)) = \max\{0, 1 - yf(x)\}$.

The Representer Theorem (Wahba, 1990; Schölkopf, Herbrich, and Smola, 2001) shows that, under quite general conditions, the solution found by minimising the regularised risk has the form

$$f^*(\cdot) = \sum_{i=1}^{n} c_i k(x_i, \cdot)$$

where $k(\cdot, \cdot)$ is the kernel corresponding to the inner product in $\mathcal{H}$, the so-called 'reproducing kernel' of the Hilbert space $\mathcal{H}$.

Using hinge loss and making use of our knowledge about the form of the solution, the problem of minimising the regularised risk can be transformed into the so-called 'primal' optimisation problem of soft-margin support vector machines:

$$\begin{array}{lll} \min\limits_{c\in\mathbb{R}^n} & \dfrac{C}{n}\displaystyle\sum_{i=1}^{n}\xi_i + c^\top K c & \\ \text{subject to:} & y_i K_{i.} c \geq 1 - \xi_i & i = 1, \ldots n \\ & \xi_i \geq 0 & i = 1, \ldots n \end{array}$$

where $K_{ij} = k(x_i, x_j)$. This optimisation problem is convex as the kernel function $k(\cdot, \cdot)$, and thus also the Hessian matrix of the objective function $R_{\text{svm}}[\cdot]$ of the above optimisation problem, is positive definite. Therefore any solution to $\frac{d}{dc} R_{\text{svm}}[c] = (0, 0, \ldots 0)^\top$ is a globally optimal solution.

2.3. *Kernels on vectors*

Before investigating kernels for structured data in detail, the traditionally used kernels (on vector spaces) are briefly reviewed in this section. Let $x, x' \in \mathbb{R}^n$ and let $\langle \cdot, \cdot \rangle$ denote the scalar product in $\mathbb{R}^n$. Apart from the linear kernel

$$k(x, x') = \langle x, x' \rangle$$

and the normalised linear kernel

$$k(x, x') = \frac{\langle x', x' \rangle}{\sqrt{\|x\| \|x'\|}}$$

the two most frequently used kernels on vector spaces are the polynomial kernel and the Gaussian RBF kernel. Given two parameters $l \in \mathbb{R}$, $p \in \mathbb{N}^+$ the polynomial kernel is defined as:

$$k(x, x') = (\langle x, x' \rangle + l)^p$$

The intuition behind this kernel definition is that it is often useful to construct new features as products of original features. This way, for example, the XOR problem can be turned into a linearly separable problem. The parameter p is the maximal order of monomials making up the new feature space, while l can be used as a bias towards lower-order monomials—for instance, if $l = 0$ the feature space consists only of monomials of order p of the original features.

Given the parameter γ, the Gaussian RBF kernel is defined as:

$$k(x, x') = e^{-\gamma \|x - x'\|^2}$$

Using this kernel function in a support vector machine can be seen as using a radial basis function network with Gaussian kernels centred at the support vectors. The images of the points from the vector space $\mathbb{R}^n$ under the map $\phi : \mathbb{R}^n \to \mathcal{H}$ with $k(x, x') = \langle \phi(x), \phi(x') \rangle$ lie all on the surface of a hyperball in the Hilbert space $\mathcal{H}$. No two images are orthogonal and any set of images is linearly independent.

The kernels on vectors described above can be found in similar form in Section 4.3 where we discuss the kernel modifiers used to adapt the default kernel for basic terms to actual problem domains. The following kernel can be used with discrete data. The matching kernel $k_\delta : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is defined as

$$k_\delta(x, x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$

The image of any element of $\mathcal{X}$ under the map $\phi : \mathcal{X} \to \mathcal{H}$ with $k_\delta(x, x') = \langle \phi(x), \phi(x') \rangle$ is a vector orthogonal to all other images of elements of $\mathcal{X}$ under ϕ.

2.4. Kernels for structured data

The best-known kernel for representation spaces that are not mere attribute-value tuples is the convolution kernel proposed by Haussler (1999). The basic idea of convolution kernels is that the semantics of composite objects can often be captured by a relation R between the object and its parts. The kernel on the object is composed of kernels defined on different parts. Let $x, x' \in \mathcal{X}$ be the objects and $\vec{x}, \vec{x}' \in \mathcal{X}_1 \times \cdots \times \mathcal{X}_D$ be tuples of parts of these objects. Given the relation $R : (\mathcal{X}_1 \times \cdots \times \mathcal{X}_D) \times \mathcal{X}$ we can define the decomposition R^{-1} as $R^{-1}(x) = \{\vec{x} : R(\vec{x}, x)\}$. With positive definite kernels $k_d : \mathcal{X}_d \times \mathcal{X}_d \to \mathbb{R}$ the convolution kernel is defined as

$$k_{\text{conv}}(x, x') = \sum_{\vec{x} \in R^{-1}(x), \vec{x}' \in R^{-1}(x')} \prod_{d=1}^{D} k_d(x_d, x'_d)$$

The term 'convolution kernel' refers to a class of kernels that can be formulated in the above way. The advantage of convolution kernels is that they are very general and can be applied in many different situations. However, because of that generality, they require a significant amount of work to adapt them to a specific problem, which makes choosing R in real-world applications a non-trivial task.

There are other kernel definitions for structured data in the literature; however, these usually focus on a very restricted syntax and are fairly domain-specific. Examples are string and tree kernels. Traditionally, string kernels (Lodhi et al., 2002) have focused on applications in text mining and measure similarity of two strings by the number of common (not necessarily contiguous) substrings. However, other string kernels have been defined for different domains, e.g., recognition of translation initiation sites in DNA and mRNA sequences (Zien et al., 2000). Tree kernels (Collins and Duffy, 2002) can be applied to ordered trees where the number of children of a node is determined by the label of the node. They compute the similarity of trees based on their common subtrees. Tree kernels have been applied in natural language processing tasks. Kernels that can be applied to graphs have recently been introduced in Gärtner (2002), Kashima and Inokuchi (2002) and Gärtner, Flach, and Wrobel (2003).

For an extensive overview of these and other kernels on structured data, the reader is referred to Gärtner (2003).

3. Knowledge representation

For a syntax-driven kernel definition, one needs a knowledge representation formalism that is able to accurately and naturally model the underlying semantics of the data. The knowledge representation formalism we use is based on the principles of using a typed syntax and representing individuals as (closed) terms. The theory behind this knowledge representation formalism can be found in Lloyd (2003) and a brief outline is given in this section. The typed syntax is important for pruning search spaces and for modelling the semantics of the data as closely as possible in a human- and machine-readable form.

The individuals-as-terms representation is a natural generalisation of the attribute-value representation as it collects all information about an individual in a single term.

The setting is a typed, higher-order logic that provides a variety of important data types, including sets, multisets, and graphs for representing individuals. The logic is based on Church's simple theory of types (Church, 1940) with several extensions. First, we assume there is given a set of type constructors $\mathfrak{T}$ of various arities. Included in $\mathfrak{T}$ is the constructor Ω of arity 0. The domain corresponding to Ω is the set containing just *True* and *False*, that is, the boolean values. The *types* of the logic are expressions built up from the set of type constructors and a set of parameters (that is, type variables), using the symbol $\rightarrow$ (for function types) and $\times$ (for product types). For example, there is a type constructor *List* used to provide the list types. Thus, if α is a type, then *List* α is the type of lists whose elements have type α. A closed type is a type not containing any parameters, the set of all closed types is denoted by $\mathfrak{S}^c$. Standard types include *Nat* (the type of natural numbers).

There is also a set $\mathfrak{C}$ of constants of various types. Included in $\mathfrak{C}$ are $\top$ (true) and $\bot$ (false). Two different kinds of constants, *data constructors* and *functions*, are distinguished. In a knowledge representation context, data constructors are used to represent individuals. In a programming language context, data constructors are used to construct data values. (Data constructors are called functors in Prolog.) In contrast, functions are used to compute on data values; functions have definitions while data constructors do not. In the semantics for the logic, the data constructors are used to construct models (cf. Herbrand models for Prolog). A *signature* is the declared type of a constant. For example, the empty list constructor [] has signature *List a*, where *a* is a parameter and *List* is a type constructor. The list constructor # (usually written infix) has signature $a \rightarrow List\ a \rightarrow List\ a$.[1] Thus # expects two arguments, an element of type α and a list of type *List* α, and produces a new list of type *List* α. If a constant C has signature α, we denote this by $C : \alpha$.

The *terms* of the logic are the terms of the typed λ-calculus, which are formed in the usual way by abstraction, tupling, and application from constants in $\mathfrak{C}$ and a set of variables. $\mathfrak{L}$ denotes the set of all terms (obtained from a particular alphabet). A term of type Ω is called a *formula*. A function whose codomain type is Ω is called a *predicate*. In the logic, one can introduce the usual connectives and quantifiers as functions of appropriate types. Thus the connectives conjunction, $\wedge$, and disjunction, $\vee$, are functions of type $\Omega \rightarrow \Omega \rightarrow \Omega$. In addition, if t is of type Ω, the abstraction $\lambda x.t$ is written $\{x \mid t\}$ to emphasise its intended meaning as a set. There is also a tuple-forming notation $(\ldots)$. Thus, if $t_1, \ldots, t_n$ are terms of type $\tau_1, \ldots, \tau_n$, respectively, then $(t_1, \ldots, t_n)$ is a term of type $\tau_1 \times \cdots \times \tau_n$.

Now we come to the key definition of basic terms. Intuitively, basic terms represent the individuals that are the subject of learning (in Prolog, these would be the ground terms). Basic terms fall into one of three kinds: those that represent individuals that are lists, trees, and so on; those that represent sets, multisets, and so on; and those that represent tuples. The second kind are abstractions. For example, the basic term representing the set $\{1, 2\}$ is

$$\lambda x.\mathit{if}\ x = 1\ \mathit{then}\ \top\ \mathit{else\ if}\ x = 2\ \mathit{then}\ \top\ \mathit{else}\ \bot,$$

and

$$\lambda x.\mathit{if}\ x = A\ \mathit{then}\ 42\ \mathit{else\ if}\ x = B\ \mathit{then}\ 21\ \mathit{else}\ 0$$

is the representation of the multiset with 42 occurrences of A and 21 occurrences of B (and nothing else). Thus we adopt abstractions of the form

$$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0$$

to represent (extensional) sets, multisets, and so on. The term s_0 here is called a default term and for the case of sets is $\bot$ and for multisets is 0. Generally, one can define default terms for each (closed) type. The set of default terms is denoted by $\mathfrak{D}$ (full details on default terms are given in Lloyd (2003)).

Definition 3.1 (Basic terms). The set of *basic terms*, $\mathfrak{B}$, is defined inductively as follows.

1. If C is a data constructor having signature $\sigma_1 \to \cdots \to \sigma_n \to (T\ a_1 \ldots a_k), t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$, and t is $C\ t_1 \ldots t_n \in \mathfrak{L}$, then $t \in \mathfrak{B}$.
2. If $t_1, \ldots, t_n \in \mathfrak{B}$, $s_1, \ldots, s_n \in \mathfrak{B}$ $(n \geq 0)$, $s_0 \in \mathfrak{D}$ and t is

$$\lambda x.\mathit{if}\ x = t_1\ \mathit{then}\ s_1\ \mathit{else}\ \ldots\ \mathit{if}\ x = t_n\ \mathit{then}\ s_n\ \mathit{else}\ s_0 \in \mathfrak{L},$$

   then $t \in \mathfrak{B}$.
3. If $t_1, \ldots, t_n \in \mathfrak{B}$ $(n \geq 0)$ and t is $(t_1, \ldots, t_n) \in \mathfrak{L}$, then $t \in \mathfrak{B}$.

Part 1 of the definition of the set of basic terms states, in particular, that individual natural numbers, integers, and so on, are basic terms. Also a term formed by applying a data constructor to (all of) its arguments, each of which is a basic term, is a basic term. Consider again lists formed using the data constructors [] having signature *List* a, and # having signature $a \to \mathit{List}\ a \to \mathit{List}\ a$. Then $A\#B\#C\#[]$ is the basic term of type *List* α representing the list $[A, B, C]$, where A, B, and C are constants having signature α. Basic terms coming from Part 1 of the definition are called *basic structures* and always have a type of the form $T\alpha_1 \ldots \alpha_n$.

The abstractions formed in Part 2 of the definition are "almost constant" abstractions since they take the default term s_0 as value for all except a finite number of points in the domain. They are called *basic abstractions* and always have a type of the form $\beta \to \gamma$. This class of abstractions includes useful data types such as (finite) sets and multisets (assuming $\bot$ and 0 are default terms). More generally, basic abstractions can be regarded as lookup tables, with s_0 as the value for items not in the table. In fact, the precise definition of basic terms in Lloyd (2003) is a little more complicated in that, in the definition of basic abstractions, $t_1, \ldots, t_n$ are ordered and $s_1, \ldots, s_n$ cannot be default terms. These conditions avoid redundant representations of abstractions.

Part 3 of the definition of basic terms just states that one can form a tuple from basic terms and obtain a basic term. These terms are called *basic tuples* and always have a type of the form $\alpha_1 \times \cdots \times \alpha_n$.

Compared with Prolog, our knowledge representation offers a type system which can be used to express the structure of the hypothesis space and thus acts as a declarative bias. The other important extension are the abstractions, which allow us to use genuine sets

and multisets. In fact, Prolog only has data constructors (functors), which are also used to emulate tuples.

It will be convenient to gather together all basic terms that have a type more general than some specific closed type. In this definition, if α and β are types, then α is *more general than* β if there exists a type substitution ξ such that $\beta = \alpha\xi$.

Definition 3.2 (Basic terms of a given type). For each $\alpha \in \mathfrak{S}^c$, define $\mathfrak{B}_\alpha = \{t \in \mathfrak{B} \mid t$ has type more general than $\alpha\}$.

The intuitive meaning of $\mathfrak{B}_\alpha$ is that it is the set of terms representing individuals of type α.

For use in the definition of a kernel, we introduce some notation. If $s \in \mathfrak{B}_{\beta\to\gamma}$ and $t \in \mathfrak{B}_\beta$, then $V(s\ t)$ denotes the "value" returned when s is applied to t. (The precise definition is in Lloyd (2003).) For example, if s is $\lambda x.$*if* $x = A$ *then* 42 *else if* $x = B$ *then* 21 *else* 0 and t is A, then $V(s\ t) = 42$. Also, if $u \in \mathfrak{B}_{\beta\to\gamma}$, the *support* of u, denoted supp(u), is the set $\{v \in \mathfrak{B}_\beta \mid V(u\ v) \notin \mathfrak{D}\}$. Thus, for the s above, supp(s) $= \{A, B\}$.

As an example of the use of the formalism, for (directed) graphs, there is a type constructor *Graph* such that the type of a graph is *Graph* $\nu\ \varepsilon$, where ν is the type of information in the vertices and ε is the type of information in the edges. *Graph* is defined by

$$\mathit{Graph}\ \nu\ \varepsilon = \{\mathit{Label} \times \nu\} \times \{(\mathit{Label} \times \mathit{Label}) \times \varepsilon\},$$

where *Label* is the type of labels. Note that this definition corresponds closely to the mathematical definition of a graph: each vertex is uniquely labelled and each edge is uniquely labelled by the ordered pair of labels of the vertices it connects.

4. Kernels for basic terms

Having introduced kernels (in Section 2) and our knowledge representation formalism (in Section 3), we are now ready to define kernels for basic terms. In Section 4.1 we define a default kernel on basic terms. In Section 4.2 we show that the kernel is positive definite. In Section 4.3 we show how to adapt the default kernel to match more closely the domain under investigation.

4.1. Default kernels for basic terms

Our definition of a kernel on basic terms assumes the existence of kernels on the various sets of data constructors. More precisely, for each type constructor $T \in \mathfrak{T}$, κ_T is assumed to be a positive definite kernel on the set of data constructors associated with T. For example, for the type constructor *Nat*, κ_{Nat} could be the *product kernel* defined by $\kappa_{\mathrm{Nat}}(n, m) = nm$. For other type constructors, say M, the matching kernel could be used, i.e., $\kappa_M(x, x') = k_\delta(x, x')$.

Definition 4.1 (Default kernel for basic terms). The function $k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ is defined inductively on the structure of terms in $\mathfrak{B}$ as follows.

1. If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = T\ \alpha_1 \ldots \alpha_k$, for some $T, \alpha_1, \ldots, \alpha_k$, then

$$k(s,t) = \begin{cases} \kappa_T(C, D) & \text{if } C \neq D \\ \kappa_T(C, C) + \sum_{i=1}^{n} k(s_i, t_i) & \text{otherwise} \end{cases}$$

   where s is $C\ s_1 \ldots s_n$ and t is $D\ t_1 \ldots t_m$.
2. If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \beta \to \gamma$, for some β, γ, then

$$k(s,t) = \sum_{\substack{u \in \text{supp}(s) \\ v \in \text{supp}(t)}} k(V(s\ u), V(t\ v)) \cdot k(u, v).$$

3. If $s, t \in \mathfrak{B}_\alpha$, where $\alpha = \alpha_1 \times \cdots \times \alpha_n$, for some $\alpha_1, \ldots, \alpha_n$, then

$$k(s,t) = \sum_{i=1}^{n} k(s_i, t_i),$$

   where s is $(s_1, \ldots, s_n)$ and t is $(t_1, \ldots, t_n)$.
4. If there does not exist $\alpha \in \mathfrak{S}^c$ such that $s, t \in \mathfrak{B}_\alpha$, then $k(s,t) = 0$.

Definition 4.1 generalises Definition 3.7.4 in Lloyd (2003) in that, for Part 2, the latter definition restricts attention to the case where k is the matching kernel on $\mathfrak{B}_\beta$ so that

$$k(s,t) = \sum_{u \in \text{supp}(s) \cap \text{supp}(t)} k(V(s\ u), V(t\ u)),$$

for $s, t \in \mathfrak{B}_{\beta \to \gamma}$. This extension is important in practice and each of the applications described in Section 6 will use it.

We proceed by giving examples and some intuition of the default kernel defined above. In the following examples we will be somewhat sloppy with our notation and sometimes identify basic terms with the lists, sets, and multisets they represent.

Example 4.1 (Default kernel on lists). Let M be a nullary type constructor and A, B, C, D: M. Let # and [] be the usual data constructors for lists. Choose κ_M and κ_{List} to be the matching kernel. Let s be the list $[A, B, C] \in \mathfrak{B}_{\text{List}\ M}$, $t = [A, D]$, and $u = [B, C]$. Then

$$\begin{aligned} k(s,t) &= \kappa_{\text{List}}((\#), (\#)) + k(A, A) + k([B, C], [D]) \\ &= 1 + \kappa_M(A, A) + \kappa_{\text{List}}((\#), (\#)) + k(B, D) + k([C], []) \\ &= 1 + 1 + 1 + \kappa_M(B, D) + \kappa_{\text{List}}((\#), []) \\ &= 3 + 0 + 0 \\ &= 3. \end{aligned}$$

Similarly, $k(s,u) = 2$ and $k(t,u) = 3$.

The intuition here is that if we use the matching kernel on the list constructors and on the elements of the lists, then we can decompose the kernel as $k(s,t) = l + m + n$ where l is the length of the shorter list, m is the number of consecutive matching elements at the start of both lists, and $n = 1$ if the lists are of the same length and 0 otherwise.

The kernel used in the above example is related to string kernels (Lodhi et al., 2002) in so far as they apply to the same kind of data. However, the underlying intuition of list/string similarity is very different. String kernels measure similarity of two strings by the number of common (not necessarily consecutive) substrings. The list kernel defined above only takes the longest common consecutive sublist at the start of the two lists into account. This is more in line with the usual ILP interpretation of lists as head-tail trees, and the kind of matching performed by anti-unification.

Example 4.2 (Default kernel on sets). Let M be a nullary type constructor and $A, B, C, D : M$. Choose κ_M and κ_Ω to be the matching kernel. Let s be the set $\{A, B, C\} \in \mathfrak{B}_{M\to\Omega}$, $t = \{A, D\}$, and $u = \{B, C\}$. Then

$$\begin{aligned} k(s,t) &= k(A,A)k(\top,\top) + k(A,D)k(\top,\top) + k(B,A)k(\top,\top) + k(B,D)k(\top,\top) \\ &\quad + k(C,A)k(\top,\top) + k(C,D)k(\top,\top) \\ &= \kappa_M(A,A) + \kappa_M(A,D) + \kappa_M(B,A) + \kappa_M(B,D) + \kappa_M(C,A) + \kappa_M(C,D) \\ &= 1 + 0 + 0 + 0 + 0 + 0 \\ &= 1. \end{aligned}$$

Similarly, $k(s,u) = 2$ and $k(t,u) = 0$.

The intuition here is that using the matching kernel for the elements of the set corresponds to computing the cardinality of the intersection of the two sets. Alternatively, this computation can be seen as the inner product of the bit-vectors representing the two sets.

Example 4.3 (Default kernel on multisets). Let M be a nullary type constructor and $A, B, C, D : M$. Choose κ_M to be the matching kernel, and κ_{Nat} to be the product kernel. Let s be $\langle A, A, B, C, C, C\rangle \in \mathfrak{B}_{M\to\text{Nat}}$ (i.e., s is the multiset containing two occurrences of A, one of B, and three of C), $t = \langle A, D, D\rangle$, and $u = \langle B, B, C, C\rangle$. Then

$$\begin{aligned} k(s,t) &= k(2,1)k(A,A) + k(2,2)k(A,D) + k(1,1)k(B,A) + k(1,2)k(B,D) \\ &\quad + k(3,1)k(C,A) + k(3,2)k(C,D) \\ &= 2 \times 1 + 4 \times 0 + 1 \times 0 + 2 \times 0 + 3 \times 0 + 6 \times 0 \\ &= 2. \end{aligned}$$

Similarly, $k(s,u) = 8$ and $k(t,u) = 0$.

The intuition here is that using the product kernel for the elements of the multiset corresponds to computing the inner product of the multiplicity vectors representing the two multisets.

These examples were kept deliberately simple in order to illustrate the main points. The kernel defined in Definition 4.1 provides much more flexibility in three important respects: (i) it allows nesting of types, such that, e.g., sets can range over objects that are themselves structured; (ii) it allows flexibility in the choice of the kernels on data constructors, such that, e.g., elements of lists can partially match as in DNA sequences; and (iii) it allows flexibility in the way in which examples are modelled in the framework, such that, e.g., a list does not have to be represented by a head-tail tree if that does not match its semantics.

4.2. Positive definiteness of the default kernel

Now we can formulate the main theoretical result of the paper.

Proposition 4.2 (*Positive definiteness*)**.** *Let $k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ be the function defined in Definition* 4.1. *For each $\alpha \in \mathfrak{S}^c$, it holds that k is a positive definite kernel on $\mathfrak{B}_\alpha$ if the kernels κ_T on the data constructors associated with the same type constructor T are positive definite.*

The full inductive proof of this Proposition is given in the Appendix. A similar result for a less general kernel function can be found in Lloyd (2003). The key idea of the proof is to base the induction on a 'bottom-up' definition of $\mathfrak{B}$. Here is the relevant definition.

Definition 4.3. Define $\{\mathfrak{B}_m\}_{m\in\mathbb{N}}$ inductively as follows.

$$
\begin{aligned}
\mathfrak{B}_0 &= \{C \mid C \text{ is a data constructor of arity } 0\} \\
\mathfrak{B}_{m+1} &= \{C\ t_1 \ldots t_n \in \mathfrak{L} \mid C \text{ is a data constructor of arity } n \text{ and} \\
&\qquad\qquad t_1, \ldots, t_n \in \mathfrak{B}_m (n \geq 0)\} \\
&\quad \cup \{\lambda x.\text{if } x = t_1 \text{ then } s_1 \text{ else} \ldots \text{if } x = t_n \text{ then } s_n \text{ else } s_0 \in \mathfrak{L} \mid \\
&\qquad\qquad t_1, \ldots, t_n \in \mathfrak{B}_m, s_1, \ldots, s_n \in \mathfrak{B}_m, \text{ and } s_0 \in \mathfrak{D}\} \\
&\quad \cup \{(t_1, \ldots, t_n) \in \mathfrak{L} \mid t_1, \ldots, t_n \in \mathfrak{B}_m\}.
\end{aligned}
$$

One can prove that $\mathfrak{B}_m \subseteq \mathfrak{B}_{m+1}$, for $m \in \mathbb{N}$, and that $\mathfrak{B} = \bigcup_{m\in\mathbb{N}} \mathfrak{B}_m$.

The intuitive outline of the proof of Proposition 4.2 is as follows: First, assume that those kernels occurring on the right-hand side in Definition 4.1 are positive definite. Then the positive definiteness of the (left-hand side) kernel follows from the closure properties of the class of positive definite kernels. The kernel on basic structures is positive definite because of closure under sum, zero extension, and direct sum, and because the kernels defined on the data constructors are assumed to be positive definite. The kernel on basic abstractions is positive definite as the function *supp* returns a finite set, and kernels are closed under zero extension, sum, and tensor product. The kernel on basic tuples is positive definite because of closure under direct sum.

4.3. Specifying kernels

The kernel defined in the previous section closely follows the type structure of the individuals that are used for learning. As indicated, the kernel assumes the existence of atomic kernels for all data constructors used. These kernels can be the product kernel for numbers, the matching kernel which just checks whether the two constructors are the same, or a user-defined kernel. In addition, kernel modifiers can be used to customise the kernel definition to the domain at hand. In this section we first describe some commonly used kernel modifiers. After that, we suggest how atomic kernels and kernel modifiers could be specified by an extension of the Haskell language (Jones and Hughes, 1998).

To incorporate domain knowledge into the kernel definition, it will frequently be necessary to modify the default kernels for a type. Below we formally describe these modifications in terms of a function $\kappa_{\text{modifier}} : \mathcal{P} \to (\mathcal{X} \times \mathcal{X} \to \mathbb{R}) \to (\mathcal{X} \times \mathcal{X} \to \mathbb{R})$ that—given a modifier and its parameters (an element of the parameter space $\mathcal{P}$)—maps any kernel to the modified kernel. For these modifiers, several choices are offered.

By default, no modifier is used, i.e.,

$$\kappa_{\text{default}}(k)(x, x') = k(x, x').$$

Instead, a polynomial version of the default kernel can be used:

$$\kappa_{\text{polynomial}}(p, l)(k)(x, x') = (k(x, x') + l)^p. \qquad (l \geq 0,\ \ p \in \mathbb{Z}^+)$$

Or a Gaussian version:

$$\kappa_{\text{gaussian}}(\gamma)(k)(x, x') = e^{-\gamma[k(x,x)-2k(x,x')+k(x',x')]}. \qquad (\gamma > 0)$$

Another frequently used modification is the normalisation kernel:

$$\kappa_{\text{normalised}}(k)(x, x') = \frac{k(x, x')}{\sqrt{k(x, x)k(x', x')}}.$$

Here $\sqrt{k(x, x)}$ is the norm of x in feature space, and thus the normalisation kernel is equal to the cosine of the angle between x and x' in feature space. Other modifiers can be defined by the user, using the syntax below.

We suggest that kernels be defined directly on the type structure (specifying the structure of the domain and the declarative bias). We introduce our suggested kernel definition syntax by means of an example: the East/West challenge (Michie et al., 1994).

```
eastbound :: Train -> Bool
type Train = Car -> Bool with modifier gaussian 0.1
type Car = (Shape,Length,Roof,Wheels,Load)
data Shape = Rectangle | Oval
```

```
data Length = Long | Short
data Roof = Flat | Peaked | None with kernel roofK
type Wheels = Int with kernel discreteKernel
type Load = (LShape,LNumber)
data LShape = Rectangle | Circle | Triangle
type LNumber = Int
```

The first line declares the learning target `eastbound` as a mapping from trains to the booleans. A train is a set of cars, and a car is a 5-tuple describing its shape, its length, its roof, its number of wheels, and its load. All of these are specified by data constructors except the load, which itself is a pair of data constructors describing the shape and number of loads.

The `with` keyword describes a property of a type, in this case kernels and kernel modifiers. The above declarations state that on trains we use a Gaussian kernel modifier with bandwidth $\gamma = 0.1$. By default, for `Shape`, `Length` and `LShape` the matching kernel is used, while for `LNumber` the product kernel is used. The default kernel is overridden for `Wheels`, which is defined as an integer but uses the matching kernel instead. Finally, `Roof` has been endowed with a user-defined atomic kernel which could be defined as follows:

```
roofK :: Roof -> Roof -> Real
roofK x x = 1
roofK Flat Peaked = 0.5
roofK Peaked Flat = 0.5
roofK x y = 0
```

This kernel counts 1 for identical roofs, 0.5 for matching flat against peaked roofs, and 0 in all other cases (i.e., whenever one car is open and the other is closed).

Finally, the normalisation modifier could be implemented as the following higher-order function:

```
normalised :: (t->t->Real) -> t -> t -> Real
normalised k x y = (k x y) / sqrt((k x x)*(k y y))
```

In this section we have presented a kernel for structured data that closely follows the syntactic structure of individuals as expressed by their higher-order type signature. The default setting assumes the product kernel for numbers and the matching kernel for symbols, but this can be overridden to match the semantics of the data more closely. The approach is very general in that it can be used with any kernel method. Furthermore, it is straightforward to transform a positive definite kernel into a pseudo-metric that can be used with arbitrary distance-based methods, as we discuss in the next section. Therefore, our work also contributes to the growing body of work on distance-based ILP.

5. Distances for basic terms

In this section we will briefly review the relation between kernel functions and distances satisfying the metric conditions. While technically straightforward, this relation is practically important as it extends the applicability of kernel functions to distance-based methods such as k-nearest neighbour. The main insight is that the inner product in feature space, as calculated by a positive definite kernel, can be used to define Euclidean distance in feature space as the norm of the vector $\phi(x)-\phi(x')$, i.e., $\sqrt{\langle \phi(x) - \phi(x'), \phi(x) - \phi(x') \rangle}$. This leads to the following definition.

Definition 5.1 (Distances from kernels). Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel on $\mathcal{X}$. The *distance measure induced by k* is defined as

$$d_k(x, x') = \sqrt{k(x, x) - 2k(x, x') + k(x', x')}$$

If k is a valid kernel then d_k is well-behaved in that it satisfies the conditions of a pseudo-metric: (1) $d(x, z)+d(z, x') \geq d(x, x')$ (triangle inequality), (2) $d(x, x') = d(x', x)$ (symmetry), and (3) $x = x' \Rightarrow d(x, x') = 0$. (For a metric, this latter implication is strengthened to an equivalence.)

Proposition 5.2 (*Valid kernels induce pseudo-metrics*)**.** *Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel on $\mathcal{X}$, and let $d_k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be the distance induced by k. If k is positive definite, then d_k is a pseudo-metric.*

We omit the proof as this is a basic result from linear algebra.

Thus, we can use the embedding in a linear feature space provided by a kernel to define a pseudo-metric for structured data expressed as basic terms.

Corollary 5.3 (*Default distance for basic terms*)**.** *Let $k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ be the function defined in Definition* 4.1*, and let $d_k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ be the distance induced by k. If the kernels κ_T on the data constructors associated with the same type constructor T are positive definite, then d_k is a pseudo-metric on $\mathfrak{B}_\alpha$.*

We proceed by giving a few simple examples of our default distance for basic terms.

Example 5.1 (Default distance on lists). We continue Example 4.1, with $s = [A, B, C]$, $t = [A, D]$, and $u = [B, C]$. We have $k(s, s) = 7$, $k(t, t) = 5$, and $k(u, u) = 5$. Then, $d_k(s, t) = \sqrt{k(s, s) - 2k(s, t) + k(t, t)} = \sqrt{7 - 6 + 5} = 2.45$, $d_k(s, u) = \sqrt{7 - 4 + 5} = 2.83$, and $d_k(t, u) = \sqrt{5 - 6 + 5} = 2$.

As an intuition, when using the matching kernel on the list constructors and on the elements of the lists, then $k(s, s) = 2|s| + 1$ where $|s|$ denotes the length of list s. Earlier we saw that with the matching kernel $k(s, t) = l + m + n$ where l is the length of the shorter list, m is the number of consecutive matching elements when the lists are aligned from the head,

and $n = 1$ if the lists are of the same length and 0 otherwise. Putting the two together, we have that $d_k(s,t) = \sqrt{2(m' + n')}$ where m' is the number of non-matching elements in the longer list, and $n' = 0$ if the lists are of the same length and 1 otherwise.

Example 5.2 (Default distance on sets). We continue Example 4.2, with $s = \{A, B, C\}$, $t = \{A, D\}$, and $u = \{B, C\}$. We have $k(s,s) = 3$, $k(t,t) = 2$, and $k(u,u) = 2$. Then, $d_k(s,t) = \sqrt{3 - 2 + 2} = 1.73$, $d_k(s,u) = \sqrt{3 - 4 + 2} = 1$, and $d_k(t,u) = \sqrt{2 - 0 + 2} = 2$.

Earlier we saw that with the matching kernel we get $k(s,t) = |s \cap t|$ for sets s, t, and thus $d_k(s,t) = \sqrt{|s| - 2|s \cap t| + |t|}$, i.e., the square root of the cardinality of the symmetric difference of s and t.

Example 5.3 (Default distance on multisets). We continue Example 4.3, with $s = \langle A, A, B, C, C, C\rangle$, $t = \langle A, D, D\rangle$, and $u = \langle B, B, C, C\rangle$. We have $k(s,s) = 14$, $k(t,t) = 5$, and $k(u,u) = 8$. Then, $d_k(s,t) = \sqrt{14 - 4 + 5} = 3.87$, $d_k(s,u) = \sqrt{14 - 16 + 8} = 2.45$, and $d_k(t,u) = \sqrt{5 - 0 + 8} = 3.61$.

It should be noted that the distance measure inherits the flexibility of the default kernel for basic terms in that types can be nested and the kernels on the data constructors can be adapted to the domain. Furthermore, we can use kernel modifiers such as the ones discussed in Section 4.3.

Distance functions on discrete structures have been investigated for some time. However, for most distance functions defined in the literature the metric properties do not hold. For example, one well-known distance function on discrete structures is the RIBL distance, described in Horváth, Wrobel, and Bohnebeck (2001), for which neither symmetry nor the triangle inequality hold. Specifically, symmetry is violated if two sets have the same cardinality. In Section 6.3 we show that a 1-nearest neighbour classifier using our default metric considerably outperforms RIBL on the diterpene data set, on which RIBL achieved the best result published so far.

Several other distances are summarised in Horváth, Wrobel, and Bohnebeck (2001) and Ramon and Bruynooghe (2001) where it is shown that most distances suggested so far are either non-metric or unsuited for real-world problems. An exception to this is Ramon and Bruynooghe (2001). The distance function suggested there (the RB distance) is based on the idea of computing a legal flow of maximal flow value and minimal cost in a weighted bipartite graph. It can be shown that this distance satisfies all properties of a metric. It has (to the best of our knowledge) not yet been shown whether for the RB distance there exists a positive definite kernel that induces it. On the other hand, the RB distance is more general than ours in that it allows non-ground terms.

6. Applications and experiments

In this section, we empirically investigate the appropriateness of our kernel definitions on a variety of domains. The implementation and application of most algorithms mentioned

below has been simplified by using the Weka data mining toolkit (Witten and Frank, 2000). In the tables presented in this section we will use the acronym 'KeS' (kernel for structured data) to refer to a support vector machine using our kernel for basic terms, and the acronym 'DeS' (distance for structured data) to refer to a nearest neighbour algorithm using our distance for basic terms.

6.1. East/West challenge

We performed some experiments with the East/West challenge data set introduced earlier in Section 4.3. We used the default kernels for all types, i.e., the product kernel for all numbers, the matching kernel for all other atomic types, and no kernel modifiers. As this toy data set only contains 20 labelled instances, the aim of this experiment was not to achieve a high predictive accuracy but to check whether this problem can actually be separated using our default kernel. We applied a support vector machine and a 3-nearest neighbour classifier to the full data set. In both experiments, we achieved 100% training accuracy, verifying that the data is indeed separable with the default kernels.

6.2. Drug activity prediction

Multi-instance problems have been introduced under this name in Dietterich, Lathrop, and Lozano-Pérez (1997). However, similar problems and algorithms have been considered earlier, for example in pattern recognition (Keeler, Rumelhart, and Leow, 1991). Within the last couple of years, several approaches have been proposed to upgrade attribute-value learning algorithms to tackle multi-instance problems. Other approaches focused on new algorithms specifically designed for multi-instance learning.

If examples are represented by subsets of some domain $\mathcal{X}$, concepts are functions $c_{\text{set}} : 2^{\mathcal{X}} \to \{-1, +1\}$. There are $2^{2^{|\mathcal{X}|}}$ different concepts on sets. Such concepts are sometimes referred to as multi-part concepts. Multi-instance concepts are a specific kind of these concepts.

Definition 6.0. A multi-instance concept is a function $c_{\text{MI}} : 2^{\mathcal{X}} \to \{-1, +1\}$ defined as:

$$c_{\text{MI}}(X) = \begin{cases} +1 & \text{if } \exists\, x \in X : c_{\text{I}}(x) = +1 \\ -1 & \text{otherwise} \end{cases}$$

where $c_{\text{I}} : \mathcal{X} \to \{-1, +1\}$ is a concept on an instance space (often referred to as the 'underlying concept'), and $X \subseteq \mathcal{X}$ is a set.

There are $2^{|\mathcal{X}|}$ different multi-instance concepts. The difficulty in this task is not just to generalise beyond examples, but also to identify the characteristic element of each set. Any learning algorithm that sees a positive set cannot infer much about the elements of the set, except that one of its elements is positive in the underlying concept. With large sets, this information is of limited use.

A popular real-world example of a multi-instance problem is the prediction of drug activity, introduced in Dietterich, Lathrop, and Lozano-Pérez (1997). A drug is active if it binds well to enzymes or cell-surface receptors. The binding strength is determined by the shape of the drug molecule. However, most molecules can change their shape by rotating some of their internal bonds. The possible shapes of a molecule, i.e., a combinations of the angles of the rotatable bonds of the molecule, are known as conformations. A drug binds well to enzymes or cell-surface receptors if one of its conformations binds well. Thus the drug activity prediction problem is a multi-instance problem. A molecule is represented by a set of descriptions of its different conformations. The shape of each conformation is described by a feature vector where each component corresponds to the length of one ray from the origin to the molecule surface.

The musk domain introduced in Dietterich, Lathrop, and Lozano-Pérez (1997) involves predicting the strength of synthetic musk molecules. The class labels were provided by human domain experts. Two overlapping data sets are available. Musk1 contains 47 molecules labelled as 'Musk' (if the molecule is known to smell musky) and 45 labelled as 'Non-Musk'. The 92 molecules are altogether described by 476 conformations. Musk2 contains 39 'Musk' molecules and 63 'Non-Musk' molecules, described by 6598 conformations altogether. 162 uniformly distributed rays have been chosen to represent each conformation. Additionally, four further features are used that describe the position of an oxygen atom in the conformation.

The formal specification of the structure of the musk data set along with the kernel applied in Gärtner et al. (2002) is as follows:

```
type Molecule = Con -> Bool with modifier normalised
type Con = (Rays,Distance,Offset)
                    with modifier gaussian 10^-5.5
type Rays = (Real,Real,...,Real)
type Offset = (Real,Real,Real)
type Distance = Real
```

Table 1 compares the results achieved with a support vector machine using this kernel (labelled KeS) to the results reported in Andrews, Tsochantaridis, and Hofmann (2003). All results have been achieved by multiple ten-fold cross-validations.[2] The algorithms compared are EMDD (Zhang and Goldman, 2002), maxDD (Maron and Lozano-Pérez, 1998), MI-NN (Ramon and De Raedt, 2000), IAPR (Dietterich, Lathrop, and Lozano-Pérez, 1997), mi-SVM and MI-SVM (Andrews, Tsochantaridis, and Hofmann, 2003). The parameter of the Gaussian modifier is chosen by leave-one-out cross-validation within each training fold.

Table 1. Classification accuracy (in %) on Musk1 and Musk2.

	EMDD	maxDD	MI-NN	IAPR	MI-SVM	mi-SVM	KeS
1	84.8	88.0	88.9	92.4	89.0	84.0	81.0
2	84.9	84.0	82.5	89.2	85.3	78.9	85.5

As the musk data set is rather small, for other parameters better results can be achieved. For example, choosing a parameter of $10^{-5.5}$ leads to an accuracy of 86.4% on Musk1 and 88.0% on Musk2.

It has been mentioned in the literature that multi-instance problems capture most of the complexity of relational learning problems (De Raedt, 1998). This experiment demonstrates that our general approach is competitive with special-purpose algorithms applied to structured data. Furthermore, our kernel-based approach has the additional advantage that learning problems other than simple classification can also be tackled by simply changing to a different kernel method.

6.3. Structure elucidation from spectroscopic analyses

The problem of diterpene structure elucidation from ^{13}C nuclear magnetic resonance (NMR) spectra was introduced to the machine learning community in Džeroski et al. (1998). There, different algorithms were compared on a data set of 1503 diterpene ^{13}C NMR spectra. Diterpenes are compounds made up from 4 isoprene units and are thus terpenes—the general term used for oligomers of isoprene. Terpenes are the major component of essential oils found in many plants. Often these oils are biologically active or exhibit some medical properties, most of which are due to the presence of terpenes.

NMR spectroscopy is one of the most important techniques in analytical chemistry. It is used as a tool in the search for new pharmaceutical products to help in determining the structure-activity relationships of biologically active compounds. Once these have been determined, it is clear which variations of the compound do not lose the biological activity. In NMR experiments the sample is placed in an external magnetic field and the nuclei are excited by a pulse over a range of radio frequencies. The signal emitted by the nuclei as they return to equilibrium with their surrounding is analysed to obtain an NMR spectrum of radio frequencies.

In the data set considered in Džeroski et al. (1998), each spectrum is described by the frequency and multiplicity of all peaks. Depending on the number of protons connected to the carbon atom, the multiplicity of a peak is either a singulet (no proton), a doublet (one proton), a triplet (two protons), or a quartet (three protons). The formal specification of the data and the kernel is as follows:

```
type Spectrum = Frequency -> Multiplicity
type Frequency = Real with modifier gaussian 0.6
data Multiplicity = s | d | t | q | 0 with default 0
```

In addition to the multiplicities s(ingulet), d(oublet), t(riplet), and q(uartet) we introduced also the multiplicity 0 and declared it as the default data constructor of the type `Multiplicity`. The abstraction `Spectrum` then maps every frequency (every real number) that is not emitted by the molecule to 0 and every emitted frequency to the multiplicity of the corresponding carbon atom.

The data set consists of 1503 spectra of diterpenes, classified into 23 different classes according to their skeleton structure as follows (number of examples per class in brackets):

Table 2. Classification accuracy (in %) on diterpene data.

Foil	Icl	Tilde	Ribl	KeS	DeS
46.5	65.3	81.6	86.5	94.7	97.1

Trachyloban (9), Kauran (353), Beyeran (72), Atisiran (33), Ericacan (2), Gibban (13), Pimaran (155), 6,7-seco-Kauran (9), Erythoxilan (9), Spongian(10), Cassan (12), Labdan (448), Clerodan (356), Portulan (5), 5,10-seco-Clerodan (4), 8,9-seco-Labdan (6), and seven classes with only one example each.

The accuracies reported in literature range up to 86.5%, achieved by RIBL (Emde and Wettschereck, 1996). Other results were reported for FOIL (Quinlan, 1990), TILDE (Blockeel and De Raedt, 1998), and ICL (De Raedt and Van Laer, 1995). See Table 2 for details. After including some manually constructed features, 91.2% accuracy has been achieved by the best system—a 1-nearest neighbour classifier using a first-order distance (RIBL).

We applied a support vector machine (see column 'KeS') using the above presented kernel function to the diterpene data set without the manually constructed features and achieved results between 94.74% and 95.48% accuracy over a range of parameters (the parameter of the Gaussian modifier was chosen from $\{0.6, 0.06, 0.006\}$ and the default $C = 1$ complexity parameter of the SVM was used). We also applied a 1-nearest neighbour algorithm (see column 'DeS') to this domain and achieved accuracies between 97.07% and 98.07% on the same set of parameters. For the overview in Table 2 and for the following experiments, the parameter of the Gaussian modifier was fixed to 0.6.

To further strengthen our results, we performed a kernel principal component analysis of the diterpene data and plotted a ROC curve. Kernel principal component analysis (Schölkopf, Smola, and Müller, 1999) is an algorithm that finds those directions in feature space in which the data has the highest variance. To allow for a useful illustration we restricted the data to molecules with structure classes Labdan and Clerodan. Figure 1 (left) shows the projection of molecules of these two classes onto the first two principal components. It can be seen that already the first two principal directions separate the classes Labdan and Clerodan quite well.

ROC curves, introduced into the machine learning community by Provost and Fawcett (2001), are often used to investigate the performance of learning algorithms under changing conditions such as misclassification costs or class distributions. ROC analysis can be applied to any binary classifier and illustrates the classifier's tradeoff between correctly classified positive examples (y-axis) and incorrectly classified negative examples (x-axis). The lower left corner corresponds to an algorithm classifying all examples as negative and the upper right corner corresponds to an algorithm classifying all examples as positive. The area under the ROC curve is a measure of how well an algorithm behaves under changing conditions. Figure 1 (right) shows the ROC curve obtained for a 1-nearest neighbour algorithm on Labdan and Clerodan diterpenes in a leave-one-out experiment. The area under this curve is 0.9998. This is very close to the optimal area under the ROC curve of 1.0.

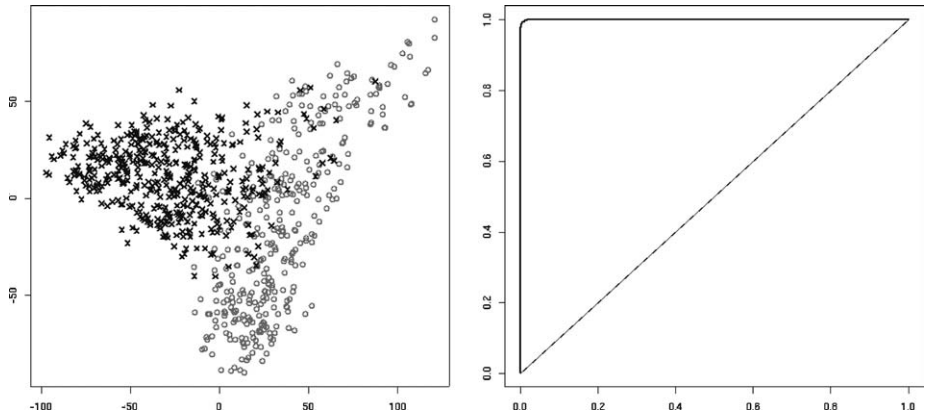

Figure 1. Left: Projection of the diterpenes with structure type Labdan and Clerodan onto their first two principal directions. Right: ROC curve for the binary classification problem of separating Labdan and Clerodan diterpenes using a 1-nearest neighbour algorithm.

6.4. Spatial clustering

The problem of clustering spatially close and thematically similar data points occurs, for example, when given demographic data about households in a city and trying to optimise facility locations given this demographic data. The location planning algorithms can usually only deal with a fairly small number of customers (less than 1000) and even for small cities the number of households easily exceeds 10,000. Therefore, several households have to be aggregated so that as little information as possible is lost. Thus the households that are aggregated have to be spatially close (so that little geographic information is lost) and similar in their demographic description (so that little demographic information is lost). The problem is to automatically find such an aggregation using an unsupervised learning algorithm.

Due to the difficulty in obtaining suitable data, we investigated this problem on a slightly smaller scale. The demographic data was already aggregated for data protection and anonymity reasons such that information is given not on a household level but on a (part of) street level. The data set describes roughly 500 points in a small German city by its geographic coordinates and 76 statistics, e.g., the number of people above or below certain age levels, the number of people above or below certain income levels, and the number of males or females living in a small area around the data point.

The simplest way to represent these data is a feature vector with 78 entries (2 for the x, y coordinates and 76 for the statistics). Drawing the results of a simple k-means algorithm on this representation clearly shows that although the spatial coordinates are taken into account, spatially compact clusters cannot be achieved. This is due to the fact that the semantics of the coordinates and the demographic statistics are different.

A better representation along with the kernel specification is as follows.

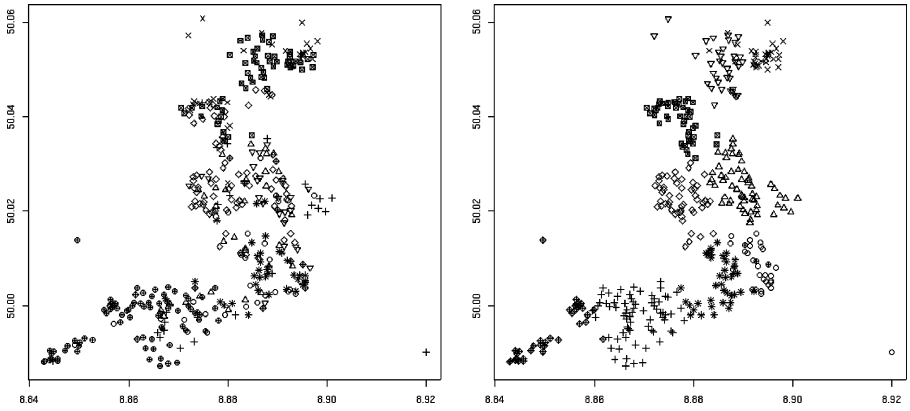

Figure 2. Plots of households spatially clustered with different parameters (left: 0.1, right: 0.02). Spatial compactness increases as the parameter is decreased.

```
type Neighbourhood = Coords -> Statistics
type Coords=(Real,Real) with modifier gaussian 0.1
type Statistics = (Real,Real,...,Real)
                              with modifier normalised
```

Here, the type `Coords` is used for the geographical coordinates and the type `Statistics` is used for the statistics of a neighbourhood of households. A neighbourhood is represented by an abstraction of type `Coords -> Statistics` that maps the coordinates of that neighbourhood to its statistics and the coordinates of all other neighbourhoods to the default term `(0,0,...,0)`. Thus each abstraction is a lookup table with a single entry corresponding to the neighbourhood represented by the abstraction. These abstractions capture the functional dependency that the coordinates determine the statistics of the neighbourhoods. It also means that the kernel on neighbourhoods multiplies the kernel on the coordinates with the kernel on the statistics. (See Part 2 of Definition 4.1.)

It is worth noting that this use of abstractions is a convenient method for getting background knowledge into a kernel. Generally, each abstraction has a single entry consisting of an individual as the item and the features of that individual as the value of that item. The features, represented as a feature vector of booleans, for example, would be constructed from the background knowledge for the application. In the application of this section, the individuals are the coordinates of neighbourhoods and the features are their statistics.

Using this representation and applying a version of the k-means algorithm[3] with the given kernel shows that the clusters are spatially compact (compactness depending on the choice of the kernel parameter). Two sample illustrations can be found in figure 2. Instances belonging to the same cluster are represented by the same symbol.[4]

7. Conclusions and future work

Bringing together kernel methods and structured data is an important direction for practical machine learning research. This requires defining a positive definite kernel on structured

data and thus embedding structured data into a linear space. In this paper we defined such a kernel, proved that it is positive definite, and showed that it works well in practice.

Our kernel definition follows a 'syntax-driven' approach, making use of a knowledge representation formalism that is able to accurately and naturally model the underlying semantics of structured data. It is based on the principles of using a typed syntax and representing individuals as (closed) terms. The typed syntax is important for pruning search spaces and for modelling as closely as possible the semantics of the data in a human- and machine-readable form. The individuals-as-terms representation is a simple and natural generalisation of the attribute-value representation and collects all information about an individual in a single term. In spite of this simplicity, the knowledge representation formalism is still powerful enough to accurately model highly structured data.

The definition of our kernel, along with the example applications presented above, show that structured data can reasonably be embedded in linear spaces. The main theoretical contribution of this paper is the proof that the kernel is positive definite on all basic terms (of the same type). The appropriateness of the kernel has been verified on some real-world domains. For instance, on the diterpene data set a support vector machine and a 1-nearest neighbour classifier using a kernel function from the framework presented in this paper improved over the best accuracy published in literature by more than 8% and 10%, respectively. This corresponds to making less than a third of the errors of the best algorithm applied to this problem so far.

Future work will consider both extending and specialising the kernel function presented above. Extending the kernel definition means, for example, considering more general abstractions than the finite lookup tables considered in this paper. Specialising the framework implies the definition of specialised kernel functions for some type constructors that have special semantics. For instance, sequences occur in text mining, in bioinformatics, in speech processing, etc., but they have a different semantics in each of these domains. Rather than a single default kernel handling sequences, we need specialised kernel definitions for each of these domains. We believe that the approach we followed in this paper, using a declarative language for kernel specifications, provides a useful framework for defining domain-specific kernels.

Appendix A: Proof of Proposition 4.2

Let $k : \mathfrak{B} \times \mathfrak{B} \to \mathbb{R}$ be the function defined in Definition 4.1. We will show that for each $\alpha \in \mathfrak{S}^c$, the following holds:

k is a positive definite kernel on $\mathfrak{B}_\alpha$ if the kernels κ_T on the data constructors associated with the same type constructor T are positive definite.

Proof: First the symmetry of k on each $\mathfrak{B}_\alpha$ is established. For each $m \in \mathbb{N}$, let $SYM(m)$ be the property:

For all $\alpha \in \mathfrak{S}^c$ and $s, t \in \mathfrak{B}_\alpha \cap \mathfrak{B}_m$, it follows that $k(s, t) = k(t, s)$.

It is shown by induction that $SYM(m)$ holds, for all $m \in \mathbb{N}$. The symmetry of k on each $\mathfrak{B}_\alpha$ follows immediately from this since, given $s, t \in \mathfrak{B}_\alpha$, there exists an m such that $s, t \in \mathfrak{B}_m$ (because $\mathfrak{B} = \bigcup_{m \in \mathbb{N}} \mathfrak{B}_m$ and $\mathfrak{B}_m \subseteq \mathfrak{B}_{m+1}$, for all $m \in \mathbb{N}$).

First it is shown that $SYM(0)$ holds. In this case, s and t are data constructors of arity 0 associated with the same type constructor T, say. By definition, $k(s, t) = \kappa_T(s, t)$ and the result follows because κ_T is symmetric.

Now assume that $SYM(m)$ holds. It is proved that $SYM(m+1)$ also holds. Thus suppose that $\alpha \in \mathfrak{S}^c$ and $s, t \in \mathfrak{B}_\alpha \cap \mathfrak{B}_{m+1}$. It has to be shown that $k(s, t) = k(t, s)$. There are three cases to consider corresponding to α having the form $T\ \alpha_1 \ldots \alpha_k$, $\beta \to \gamma$, or $\alpha_1 \times \cdots \times \alpha_m$. In each case, it is easy to see from the definition of k and the induction hypothesis that $k(s, t) = k(t, s)$. This completes the proof that k is symmetric on each $\mathfrak{B}_\alpha$.

For the remaining part of the proof, for each $m \in \mathbb{N}$, let $PD(m)$ be the property:

> For all $n \in \mathbb{Z}^+, \alpha \in \mathfrak{S}^c, t_1, \ldots, t_n \in \mathfrak{B}_\alpha \cap \mathfrak{B}_m$, and $c_1, \ldots, c_n \in \mathbb{R}$, it follows that $\sum_{i,j \in \{1,\ldots,n\}} c_i c_j k(t_i, t_j) \geq 0$.

It is shown by induction that $PD(m)$ holds, for all $m \in \mathbb{N}$. The remaining condition for positive definiteness follows immediately from this since, given $t_1, \ldots, t_n \in \mathfrak{B}_\alpha$, there exists an m such that $t_1, \ldots, t_n \in \mathfrak{B}_m$.

First it is shown that $PD(0)$ holds. In this case, each t_i is a data constructor of arity 0 associated with the same type constructor T, say. By definition, $k(t_i, t_j) = \kappa_T(t_i, t_j)$, for each i and j, and the result follows since κ_T is assumed to be positive definite.

Now assume that $PD(m)$ holds. It is proved that $PD(m+1)$ also holds. Thus suppose that $n \in \mathbb{Z}^+, \alpha \in \mathfrak{S}^c, t_1, \ldots, t_n \in \mathfrak{B}_\alpha \cap \mathfrak{B}_{m+1}$, and $c_1, \ldots, c_n \in \mathbb{R}$. It has to be shown that $\sum_{i,j \in \{1,\ldots,n\}} c_i c_j k(t_i, t_j) \geq 0$. There are three cases to consider.

1. Let $\alpha = T\ \alpha_1 \ldots \alpha_k$. Suppose that $t_i = C_i\ t_i^{(1)} \ldots t_i^{(m_i)}$, where $m_i \geq 0$, for $i = 1, \ldots, n$. Let $\mathcal{C} = \{C_i \mid i = 1, \ldots, n\}$. Then

$$\sum_{i,j \in \{1,\ldots,n\}} c_i c_j k(t_i, t_j) = \sum_{i,j \in \{1,\ldots,n\}} c_i c_j \kappa_T(C_i, C_j) + \sum_{\substack{i,j \in \{1,\ldots,n\} \\ C_i = C_j}} c_i c_j \sum_{l \in \{1,\ldots,arity(C_i)\}} k\big(t_i^{(l)}, t_j^{(l)}\big).$$

Now

$$\sum_{i,j \in \{1,\ldots,n\}} c_i c_j \kappa_T(C_i, C_j) \geq 0$$

using the fact that κ_T is a positive definite kernel on the set of data constructors associated with T. Also

$$\begin{aligned}
&\sum_{\substack{i,j\in\{1,\ldots,n\}\\ C_i=C_j}} c_i c_j \sum_{l\in\{1,\ldots,\,arity(C_i)\}} k\big(t_i^{(l)}, t_j^{(l)}\big)\\
&= \sum_{C\in\mathcal{C}} \sum_{\substack{i,j\in\{1,\ldots,n\}\\ C_i=C_j=C}} \sum_{l\in\{1,\ldots,\,arity(C)\}} c_i c_j k\big(t_i^{(l)}, t_j^{(l)}\big)\\
&= \sum_{C\in\mathcal{C}} \sum_{l\in\{1,\ldots,\,arity(C)\}} \sum_{\substack{i,j\in\{1,\ldots,n\}\\ C_i=C_j=C}} c_i c_j k\big(t_i^{(l)}, t_j^{(l)}\big)\\
&\geq 0,
\end{aligned}$$

by the induction hypothesis.

2. Let $\alpha = \beta \to \gamma$. Then

$$\begin{aligned}
&\sum_{i,j\in\{1,\ldots,n\}} c_i c_j k(t_i, t_j)\\
&= \sum_{i,j\in\{1,\ldots,n\}} c_i c_j \sum_{\substack{u\in supp(t_i)\\ v\in supp(t_j)}} k(V(t_i\ u), V(t_j\ v)) \cdot k(u, v)\\
&= \sum_{i,j\in\{1,\ldots,n\}} \sum_{\substack{u\in supp(t_i)\\ v\in supp(t_j)}} c_i c_j k(V(t_i\ u), V(t_j\ v)) \cdot k(u, v)\\
&= \sum_{\substack{(i,u),(j,v)\in\\ \{(k,w)\mid k=1,\ldots,n \text{ and } w\in supp(t_k)\}}} c_i c_j k(V(t_i\ u), V(t_j\ v)) \cdot k(u, v)\\
&\geq 0.
\end{aligned}$$

For the last step, we proceed as follows. By the induction hypothesis, k is positive definite on both $\mathfrak{B}_\beta \cap \mathfrak{B}_m$ and $\mathfrak{B}_\gamma \cap \mathfrak{B}_m$. Hence the function

$$h : ((\mathfrak{B}_\beta \cap \mathfrak{B}_m) \times (\mathfrak{B}_\gamma \cap \mathfrak{B}_m)) \times ((\mathfrak{B}_\beta \cap \mathfrak{B}_m) \times (\mathfrak{B}_\gamma \cap \mathfrak{B}_m)) \to \mathbb{R}$$

defined by

$$h\,((u, y), (v, z)) = k(u, v) \cdot k(y, z)$$

is positive definite, since h is a tensor product of positive definite kernels (Schölkopf and Smola, 2002). Now consider the set

$$\{(u, V(t_i\ u)) \mid i = 1, \ldots, n \text{ and } u \in supp(t_i)\}$$

of points in $(\mathfrak{B}_\beta \cap \mathfrak{B}_m) \times (\mathfrak{B}_\gamma \cap \mathfrak{B}_m)$ and the corresponding set of constants

$$\{c_{i,u} \mid i = 1, \ldots, n \text{ and } u \in supp(t_i)\},$$

where $c_{i,u} = c_i$, for all $i = 1, \ldots, n$ and $u \in supp(t_i)$.

3. Let $\alpha = \alpha_1 \times \cdots \times \alpha_m$. Suppose that $t_i = (t_i^{(1)}, \ldots, t_i^{(m)})$, for $i = 1, \ldots, n$. Then

$$\begin{aligned}\sum_{i,j\in\{1,\ldots,n\}} c_i c_j k(t_i, t_j) &= \sum_{i,j\in\{1,\ldots,n\}} c_i c_j \left(\sum_{l=1}^{m} k\big(t_i^{(l)}, t_j^{(l)}\big) \right) \\ &= \sum_{l=1}^{m} \sum_{i,j\in\{1,\ldots,n\}} c_i c_j k\big(t_i^{(l)}, t_j^{(l)}\big) \\ &\geq 0,\end{aligned}$$

by the induction hypothesis.

□

Acknowledgments

Research supported in part by the EU Framework V project (IST-1999-11495) *Data Mining and Decision Support for Business Competitiveness: Solomon Virtual Enterprise*, by the BMBF funded project *KogiPlan*, and by the DFG project (WR 40/2-1) *Hybride Methoden und Systemarchitekturen für heterogene Informationsräume*. The authors thank Tamás Horváth, Stefan Wrobel, and the anonymous reviewers for valuable comments.

Notes

1. This could be read as $a \times List\ a \to List\ a$.
2. Actually, KeS has been evaluated using leave-ten-out cross-validation. However, as the number of instances in both cases is roughly 100, ten-fold and leave-ten-out cross-validation are equivalent for all practical purposes.
3. Note that performing k-means in feature space requires some modifications of the algorithm. A description is beyond the scope of this paper.
4. Coloured illustrations with other parameters can be found at `http://www.ais.fraunhofer.de/~thomasg/SpatialClustering/`

References

Andrews, S., Tsochantaridis, I., & Hofmann, T. (2003). Support vector machines for multiple-instance learning. In *Advances in neural information processing systems* (Vol. 15) MIT Press.

Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68.

Ben-Hur, A., Horn, D., Siegelmann, H. T., & Vapnik, V. (2001). Support vector clustering. *Journal of Machine Learning Research*, 2, 125–137.

Blockeel, H., & De Raedt, L. (1998). Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101:1/2, 285–297.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In D. Haussler (Ed.), *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory* (pp. 144–152). ACM Press.

Church, A. (1940). A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5, 56–68.

Collins, M., & Duffy, N. (2002). Convolution kernels for natural language. In T. G. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems* (Vol. 14) MIT Press.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines (and other kernel-based learning methods).* Cambridge University Press.

De Raedt, L. (1998). Attribute value learning versus inductive logic programming: The missing links (extended abstract). In D. Page (Ed.), *Proceedings of the 8th International Conference on Inductive Logic Programming*, Vol. 1446 of *Lecture Notes in Artificial Intelligence* (pp. 1–8). Springer-Verlag.

De Raedt, L., & Van Laer, W. (1995). Inductive constraint logic. In K. Jantke, T. Shinohara, & T. Zeugmann (Eds.), *Proceedings of the 6th International Workshop on Algorithmic Learning Theory*, Vol. 997 of *LNAI*, (pp. 80–94). Springer Verlag.

Dietterich, T. G., Lathrop, R. H., & Lozano-Pérez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89:1/2, 31–71.

Džeroski, S., & Lavrač N. (Eds.) (2001). *Relational data mining*. Springer-Verlag.

Džeroski, S., Schulze-Kremer, S., Heidtke, K., Siems, K., Wettschereck, D., & Blockeel, H. (1998). Diterpene structure elucidation from ^{13}C NMR spectra with inductive logic programming. *Applied Artificial Intelligence*, 12:5, 363–383. Special Issue on First-Order Knowledge Discovery in Databases.

Emde, W., & Wettschereck, D. (1996). Relational instance-based learning. In *Proceedings of the 13th International Conference on Machine Learning* (pp. 122–130). Morgan Kaufmann.

Evgeniou, T., Pontil, M., & Poggio, T. (2000). Regularization networks and support vector machines. *Advances in Computational Mathematics*.

Gärtner, T. (2002). Exponential and geometric kernels for graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*.

Gärtner, T. (2003). A survey of kernels for structured data. *SIGKDD Explorations*.

Gärtner, T., Flach, P. A., Kowalczyk, A., & Smola, A. J. (2002). Multi-instance kernels. In C. Sammut & A. Hoffmann (Eds.), *Proceedings of the 19th International Conference on Machine Learning* (pp. 179–186). Morgan Kaufmann.

Gärtner, T., Flach, P. A., & Wrobel, S. (2003). On graph kernels: Hardness results and efficient alternatives. In *Proceedings of the 16th Annual Conference on Computational Learning Theory and the 7th Kernel Workshop*.

Haussler, D. (1999). Convolution kernels on discrete structures. Technical report, Department of Computer Science, University of California at Santa Cruz.

Horváth, T., Wrobel, S., & Bohnebeck, U. (2001). Relational instance-based learning with lists and terms. *Machine Learning*, 43:1/2, 53–80.

Jones, S. P., & Hughes J. (Eds.) (1998). *Haskell98: A Non-Strict Purely Functional Language*. Available at `http://haskell.org/`.

Kashima, H., & Inokuchi, A. (2002). Kernels for graph classification. In *ICDM Workshop on Active Mining*.

Keeler, J. D., Rumelhart, D. E., & Leow, W.-K. (1991). Integrated segmentation and recognition of hand-printed numerals. In R. Lippmann, J. Moody, & D. Touretzky (Eds.), *Advances in neural information processing systems*, Vol. 3 (pp. 557–563). Morgan Kaufmann.

Lloyd, J. W. (2003). *Logic for learning*. Springer-Verlag.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., & Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2, 419–444.

Maron, O., & Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In M. I. Jordan, M. J. Kearns, & S. A. Solla (Eds.), *Advances in neural information processing systems*, Vol. 10. MIT Press.

Michie, D., Muggleton, S., Page, D., & Srinivasan, A. (1994). To the international computing community: A new EastWest challenge. Technical report, Oxford University Computing laboratory, Oxford, UK.

Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., & Schölkopf, B. (2001). An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks, 2:2*.

Provost, F., & Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learing*, 42:3, 203–231.

Quinlan, J. (1990). Learning logical definitions from relations. *Machine Learning*, 5:3, 239–266.

Ramon, J., & Bruynooghe, M. (2001). A polynomial time computable metric between point sets. *Acta Informatica*, 37:10, 765–780.

Ramon, J., & De Raedt, L. (2000). Multi instance neural networks. In *Attribute-Value and Relational Learning: Crossing the Boundaries. A Workshop at the Seventeenth International Conference on Machine Learning (ICML-2000)*.

Schölkopf, B., Herbrich, R., & Smola, A. J. (2001). A generalized representer theorem. In *Proceedings of the 14th Annual Conference on Learning Theory*.

Schölkopf, B., & Smola, A. J. (2002). *Learning with kernels*. MIT Press.

Schölkopf, B., Smola, A. J., & Müller, K.-R. (1999). Kernel principal component analysis. In B. Schölkopf, C. Burges, & A. Smola (Eds.), *Advances in kernel methods—support vector learning* (pp. 327–352). MIT Press.

Tikhonov, A. N., & Arsenin, V. Y. (1977). *Solutions of Ill-posed problems*. W.H. Winston.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer-Verlag.

Wahba, G. (1990). *Spline Models for Observational Data*, Vol. 59 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. Philadelphia: SIAM.

Witten, I. H., & Frank, E. (2000). *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann.

Zhang, Q., & Goldman, S. (2002). EM-DD: An improved multiple-instance learning technique. In T. Dietterich, S. Becker, & Z. Ghahramani (Eds.), *Advances in neural information processing systems*, Vol. 14. MIT Press.

Zien, A., Ratsch, G., Mika, S., Schölkopf, B., Lengauer, T., & Muller, K.-R. (2000). Engineering support vector machine kernels that recognize translation initiation sites. *Bioinformatics*, 16:9, 799–807.

Received March 18, 2003
Accepted November 1, 2003
Final manuscript June 21, 2004